Title
A new evolutionary search strategy for global optimization of high-dimensional problems

Journal
Information Sciences, 181(22)

ISSN
0020-0255

Authors
Chu, W
Gao, X
Sorooshian, S

Publication Date
2011-11-15

DOI
10.1016/j.ins.2011.06.024

Information Sciences 181 (2011) 4909–4927

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A new evolutionary search strategy for global optimization of high-dimensional problems

Wei Chu *, Xiaogang Gao, Soroosh Sorooshian

Department of Civil and Environmental Engineering, University of California, Irvine, CA 92617, USA

ARTICLE INFO

Article history:
Received 24 July 2010
Received in revised form 26 April 2011
Accepted 5 June 2011
Available online 13 July 2011

Keywords:
Global optimization
High-dimensional
Principal components analysis
Shuffled complex evolution
Evolutionary strategy

ABSTRACT

Global optimization of high-dimensional problems in practical applications remains a major challenge to the research community of evolutionary computation. The weakness of randomization-based evolutionary algorithms in searching high-dimensional spaces is demonstrated in this paper. A new strategy, SP-UCI is developed to treat complexity caused by high dimensionalities. This strategy features a slope-based searching kernel and a scheme of maintaining the particle population's capability of searching over the full search space. Examinations of this strategy on a suite of sophisticated composition benchmark functions demonstrate that SP-UCI surpasses two popular algorithms, particle swarm optimizer (PSO) and differential evolution (DE), on high-dimensional problems. Experimental results also corroborate the argument that, in high-dimensional optimization, only problems with well-formative fitness landscapes are solvable, and slope-based schemes are preferable to randomization-based ones.

1. Introduction

Rapid advancements in technology and science pose new challenges to the optimization community. In real-world applications, more and more complex systems, such as power systems, protein structures, medical image registration, and financial market [4,16,19,24], are being simulated by computer models with the aid of elevated cyber infrastructures. These models have increasingly elaborate structures resulting from representing multiplex physical or conceptual processes and, hence, possess large numbers of parameters. Therefore, optimization of these models is indeed searching the solutions in high-dimensional spaces spanned by the model parameters. In high-dimensional spaces, there exist many unique difficulties and phenomena which are not present in low-dimensional problems and, therefore, plague the well-performing methods in low-dimensional spaces. For instance, "curse of dimensionality", which was first coined by Bellman [1,2], is the term used to describe the problem caused by the exponential increase in volume associated with adding extra dimensions to a mathematical space.

Furthermore, Mendes et al. [22] argued that only functions whose fitness landscapes provide clues to locations of the solutions (optima) can be called problems; other functions, such as "deceptive functions" (where the gradients lead a hill-descender/-climber away from global optima) and "random functions" (where gradients exist but are unrelated to solutions), are nonsense. This argument is more plausible when we deal with high-dimensional problems. Actually, the "clues" here mainly refer to the gradient or slope of the response surface. Fortunately, many real-world applications fall into this class [22]. Thus, algorithm developers should focus on solving problems instead of intricate benchmark functions.

* Corresponding author.
E-mail address: wchu2@uci.edu (W. Chu).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.06.024

However, although the deceptive and random functions are short of practical importance, many (probably more and more) benchmark functions are designed to represent deceptiveness and randomness [11,28]. Consequently, many algorithms or algorithm alternatives attempting to tackle these two difficulties have been developed recently. Obviously, there is only one way to achieve the success on deceptive or random functions: randomization. On deceptive or random fitness landscapes, attractive regions of global minima can only be reached if a searching particle jumps into it by chance. Therefore, based on genetic-, swarm-, annealing-, and hybrid-mechanisms, many searching algorithms heavily integrate randomization in the offspring reproduction process, such as simulated annealing (SA) [15], particle swarm optimizer (PSO) [14], differential evolution (DE) [26], covariance matrix adaptation-evolution strategy (CMA-ES) [12], and their recent modifications [3,7,8,23,25,30–32,34,35]. These algorithms demonstrate outstanding performances when applied to low-dimensional deceptive and random functions and have prevailed in recent literature. However, when problem dimensionality increases, these algorithms lose their effectiveness [11] due to the fact that the power of randomization drops geometrically. The vulnerability of randomization to high dimensionality is easy to understand and can be demonstrated by a simple example:

"Assume that the global optimum is a "quadrant" (the analogy of the 2-D quadrant in the high-dimensional space) instead of a point in an n-dimensional search space. A particle is randomly jumping in the search space. To make it simple, we define that the optimization succeeds if the particle jumps into the correct quadrant (the global optimum). Therefore, the probability of success at every step is 2^{-n}. With low-dimensional problems, the probability is acceptable if we have a large population of particles or if we can evolve particles many times, which is the case of most evolutionary optimization algorithms. However, for high-dimensional problems, the probability becomes so small that the problem cannot be solved in practice. For example, with $n = 100$, the probability is only 2^{-100}, which is much lower than the probability of winning any lottery on the Earth!".

Another major concern when applying evolutionary algorithms to real-world applications is efficiency. Unlike mathematical benchmark functions, the computation time for real-world problems can be substantial, especially in high-dimensional settings or when complex processes are involved. A successful optimization algorithm must be able to evolve in a parsimonious manner in many situations. Algorithms relying heavily on randomization sacrifice their efficiency due to the fact that the more randomness in the offspring-generating mechanism, the lower probability of producing qualified offspring to keep the population evolving. As a result, these algorithms usually require a large number of function evaluations. For many high-dimensional real-world applications, the computations last too long to be viable.

To solve high-dimensional real-world optimization problems, an algorithm that can wisely exploit response surfaces and possesses high efficiency is in great need. Motivated by this consideration, we introduce an innovative evolutionary algorithm called the shuffled complex evolution with principal components analysis–University of California at Irvine (SP-UCI). Six composition functions in [20], CF1–CF6, are used to benchmark the SP-UCI method in comparison with the PSO and DE methods. Having irregular fitness landscapes and high levels of complexity, the composition functions can sufficiently present the difficulties of high-dimensional problems. Recent studies have shown that the composition benchmark functions pose large difficulties for global optimization algorithms [20]. To test the scaling behaviors of SP-UCI, the benchmark experiments are conducted in 10, 50, 100, and 1000 dimensions. Experimental results demonstrate that the SP-UCI method exhibits consistently superior performances in high-dimensional (>50-D) settings in comparison with the PSO and DE methods, and its merits become more significant as the dimensionality increases.

The no free lunch (NFL) theorems [33] state that "for any algorithm, any elevated performance over one class of problems is offset by performance over another class". In other words, there is no such algorithm that can outperform others over all problems. As a practical example of NFL, Langdon and Poil [18] demonstrated that genetic programming (GP) can readily find simple functions (response surfaces) which suite one heuristic algorithm over the other and vice versa. Following this line of reasoning, the development of any search algorithm should be accompanied by statements describing what kind of problems it is designed for. In contrast, the practice for many algorithm developers is that they try to claim that their methods perform well in general and try to use a set of benchmark functions to support them. However, a set of benchmark functions are never sufficient to represent problems in general. Therefore, we want to state that the SP-UCI method is designed for high-dimensional and complex problems. Furthermore, the superior performance of this method on the benchmark functions substantiates that searching in high-dimensional space should rely on strategies which can cleverly and efficiently exploit fitness landscapes, such as slope-based algorithms, instead of relying heavily on randomization processes.

2. The SP-UCI method

The SP-UCI method is developed based on the shuffled complex evolution-University of Arizona (SCE-UA) method [9]. Since its debut, the SCE-UA method has been widely used in calibrating conceptual hydrological models which generally have very complicated fitness landscapes with uncountable local minima, unknown roughness, and discontinuities. Studies show that the SCE-UA method has demonstrated efficiency and effectiveness on both benchmark functions and real-world applications [10,27,29]. However, when it was designed, SCE-UA was constructed primarily for and tested on low-dimensional problems. Recently, our study [6] reveals that SCE-UA tends to malfunction on high-dimensional problems, due to "population degeneration", which will be introduced in the following paragraph.

SCE-UA employs the n-dimensional Nelder–Mead simplex (hereafter referred to as simplex) scheme as the searching and evolving kernel. The simplex scheme is an effective tool reproducing qualified offspring by estimating the deepest

descent direction in its proximity. As one of the pioneers in direct search, the simplex method has been intensively studied both experimentally and theoretically [13,17,21]. As a local search algorithm, simplex exhibits high efficiency with a convergence rate close to $O(n)$ [17]. However, as revealed in our study [6], the offspring particles reproduced through a series of simplex processes may converge into a subspace of the original search space; namely, the space spanned by searching particles has a lower dimensionality than the one of the original search space. Since then, subsequent evolutions will be restricted within the subspace and have little chance of recovering to full search in the parameter space. We refer to this phenomenon as "population degeneration", which is caused by the fact that there is no mechanism to maintain the population's capability of spanning the full search space. Actually, this deficiency lies in many direct search algorithms, where the sample population evolves by itself without any supervision on dimensionality change of the population. Population degeneration may lead to disastrous consequences, such as converging to non-stationary points or unexpected divergence [6].

In response to this deficiency, we develop a scheme which utilizes principal components analysis (PCA) to identify and search along the dimensions that are not spanned by the sample population. By integrating this scheme, the SP-UCI method guarantees that the particle population is able to search in the full space during every loop throughout the entire evolution.

In detail, the SP-UCI method incorporates four concepts that are expressed by individual algorithmic modules: (a) the complex shuffling scheme; (b) population dimensionality monitoring and restoration; (c) modified competitive complex evolution (MCCE) strategy; and (d) multinormal resampling. Modules (b) and (d) are new developments to enhance the method's performance on high-dimensional and complex problems, whereas Modules (a) and (c) are inherited from the SCE-UA method with some modifications. Each of these modules is particularly designed to account for one of the major difficulties in direct search: The complex shuffling scheme, Module (a), is powerful in exploring multimodal response surfaces; Module (b) remedies the potential population degeneration; Module (c) is a sophisticated implementation of the simplex method which drives the evolution in an efficient manner; and, finally, multinormal resampling, Module (d), helps search over rough fitness landscapes. Figs. 1–4 are the flowcharts of SP-UCI, and the details of each module are presented in Appendix A. The Matlab codes of the SP-UCI method are available upon request.

3. Test functions and experimental settings

3.1. Test functions

For the sake of testing the performance of our method in real-world applications and illustrating our arguments about searching on high-dimensional problems, we choose six recently developed novel composition functions, CF1–CF6, in [20] as benchmark functions. The following considerations make these functions preferable over widely used standard benchmark functions:

(1) These functions all have very complicated and irregular fitness landscapes; hence, they have the ability to mimic real-world problems.
(2) There are no known patterns (such as the layout of minima) hidden in these functions, which prevent the tested algorithms from taking advantage of any known properties of response surfaces.
(3) The absence of symmetry makes the complexities of these functions increase substantially with dimensionality. Therefore, they can effectively test a algorithm performance's on high-dimensional problems.
(4) They are composed of a series of popular standard functions and, therefore, can represent difficulties of a wide range of standard benchmark functions.

Generally, each of these functions is composed of ten basic test functions chosen among Sphere, Ackely, Griewank, Rastrigin, and Weierstrass functions. The selected functions are randomly located and are biased by different magnitudes to generate global minimum and several local minima. From the first to the 10th basic function, biases of $0, 100, \ldots, 900$ are added respectively. Therefore, the minimum of the first basic function is the global optimum with a value of 0, and minima of the other nine basic functions are local minima with an altitude of $100, \ldots, 900$, respectively. The minimum of the 10th basic function is set to the origin in order to trap algorithms that take advantage of global optimum at the center of the search space. Furthermore, instead of simple summation, the ten basic functions are combined with Gaussian functions to blur structures of individual functions. Details of constructions and properties of the composition functions can be found in [20].

For many popular basic test functions, high dimensionality does not affect their regularities, such as symmetric landscape and uncorrelated dimensions. Therefore, the difficulties of these functions do not increase very much in high-dimensional space. In contrast, these composition functions have very irregular fitness landscapes, as shown in Fig. 5. Even in 2-D, these plots vividly display the harshness of these functions. However, the harshness will increase greatly when dimension increases. For example, the ratio of the global minimum attractive region to the whole search region will decrease geometrically with dimensionality, and the depth of local minima and magnitude of noise will increase. Thus, this group of benchmark functions can provide a rigorous examination on an algorithm's capability to search on high-dimensional complex problems.

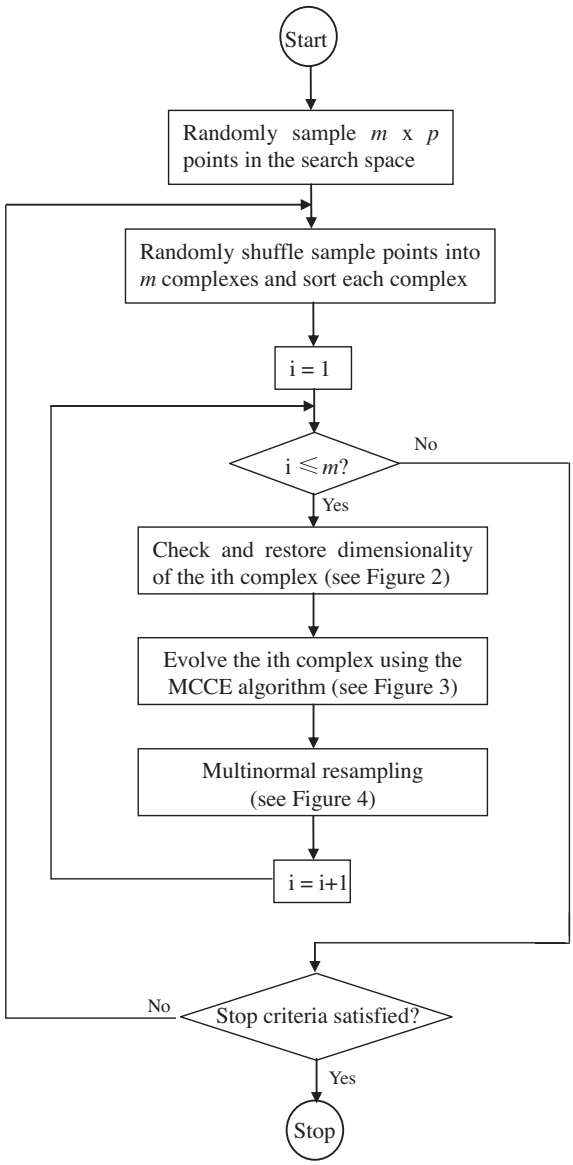

Fig. 1. The main routine of the SP-UCI method.

3.2. Experiment settings

In addition to SP-UCI, we also tested two other widely-used algorithms: particle swarm optimizer (PSO) and differential evolution (DE) (see Appendix B for brief descriptions of PSO and DE). The reason for selecting these two algorithms is two-fold: first, as two of the most popular heuristic optimization algorithms, they are good references for inter-comparisons among different studies. Second, as discussed in our previous study [6], these two algorithms, together with the SP-UCI method, represent three major types of offspring-generating schemes: DE relies heavily on randomization to enhance its robustness; SP-UCI takes the advantage of slopes of response surfaces to achieve efficiency; and PSO falls somewhere between DE and SP-UCI, combining response surface information provided by best points (in population and in history) and randomly-generated weighting vectors. Benchmarking these three algorithms will shed light on how the performances of randomization-based and slope-based searching algorithms change with dimensionality.

Liang et al. [20] reported that CMA-ES was outperformed by PSO and DE on these sophisticated composition functions. Therefore the CMA-ES method is excluded from the comparison.

3.2.1. General settings

Our experiments were conducted on 10-, 50- and 100-D. Due to computational constraints, experiments of 1000-D were only carried out for CF1. All three algorithms were run 30 times on each benchmark function. All functions had the same

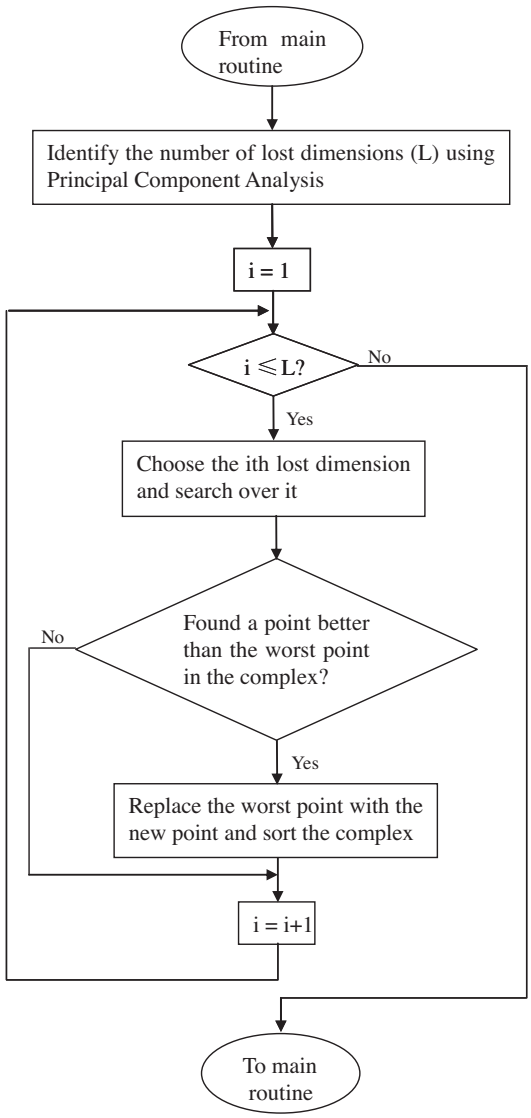

Fig. 2. The module of dimensionality monitoring and restoration.

search range of $[-5,5]$ on every dimension. Locations and magnitudes of global minima and local minima were set at the default values as in the codes provided by the authors of the composition functions.

Sample population was randomly initialized in the search range with uniform distribution. Population size was kept the same across three methods: $8 \times d + 4$, which means that there were four complexes in the SP-UCI method.

The maximum number of function evaluation was set to $10^4 \times d$, for $d = 10$, 50, 100, and $10^5 \times d$, for $d = 1000$. In addition to the maximum function evaluations, there were two additional stop criteria for SP-UCI, as specified below.

3.2.2. Algorithmic settings

SP-UCI

Two additional stop criteria: if population converges to a space of geometric size less than 10^{-6}; or if the best function value has not improved by 0.1% over the last 50 loops.

PSO

The significant coefficients: the inertia weight was held as a constant $c_0 = 0.5$. c_1 and c_2 were set at 2. V_{max} was half of the search range.

DE

The crossover constant $C = 0.5$; the scaling factor $F = 0.5$.

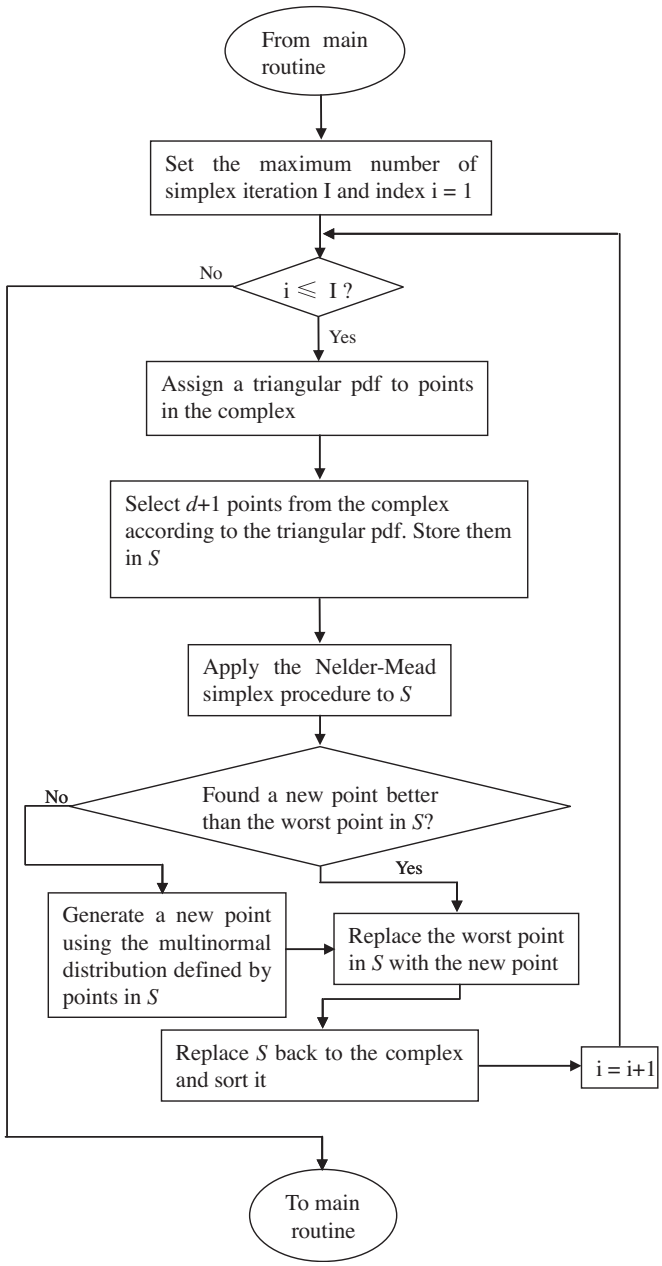

Fig. 3. The module of modified competitive complex evolution strategy.

3.2.3. Bound handling

In another recent study [5], we found that bound handling is critical to the performance of PSO. It is revealed that the widely-used random and absorbing bound-handling schemes may paralyze PSO when applied to high-dimensional problems. Therefore, in this study, we adopted a reflecting bound-handling method for all three algorithms. In this method, when a particle flies outside a bound in one of the dimensions, the bound will act like a mirror and reflect the projection of the particle's displacement.

4. Experiment results

4.1. Results description and comparison

1) 10-D problems: Fig. 6 presents the final best function values of the 30 runs of three algorithms on six benchmark functions. DE surpassed the other two on 10-D functions, and it succeeded on CF1, CF2, and CF5 in all 30 runs and on CF3 in 12 runs. On CF6, the majority of the runs of DE reached below the function value of 600. SP-UCI performed slightly

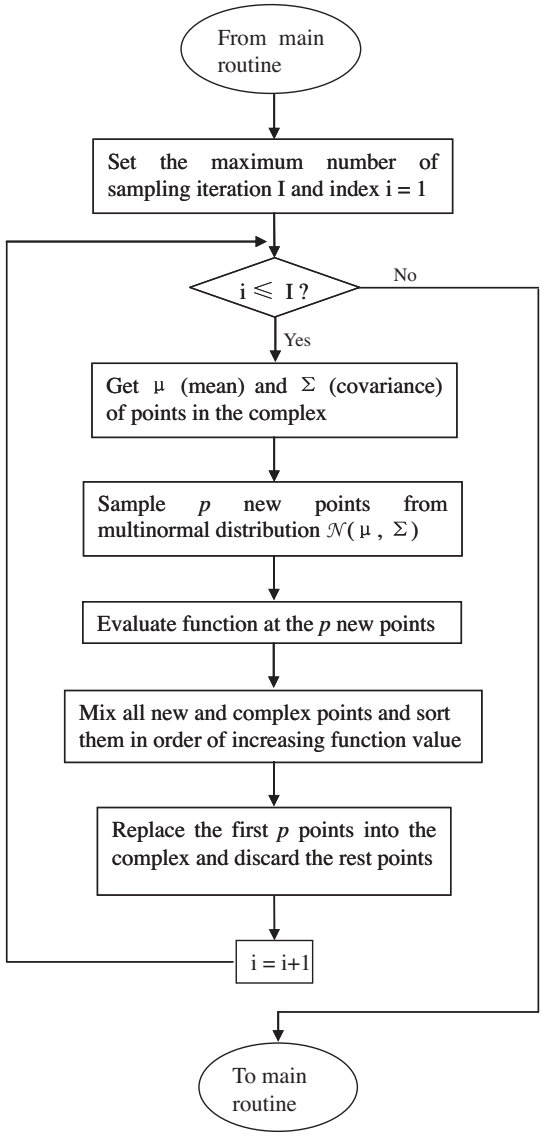

Fig. 4. The module of multinormal resampling.

better than PSO with success runs on CF1–3 and CF5. PSO only reached the global minimum on CF1 and CF5. CF4 and CF6 are too complex for any algorithm to work well. In particular, on CF4, most runs were trapped by the local minimum with a value of 600.

2) 50-D problems: results are presented in Fig. 7. DE was not outstanding anymore in this scenario, and performances of all three algorithms deteriorated. Only on CF1, most runs ended up at the global minimum, except for seven runs of PSO. On CF2, only one run of SP-UCI and one run of PSO succeeded, whereas other runs were all trapped by local minima. For CF5, most runs got very close to global minimum, but on run can overcome the roughness around to capture it. For CF6, all runs were captured by the local minima defined by a basic function with a magnitude of 900. The same thing happened to PSO and SP-UCI on CF4, and DE seems to escape this fate but not going too far.
3) 100-D problems: as shown in Fig. 8, SP-UCI outperformed the other two on every benchmark function by achieving lower or equal function values. Due to increased complexity resulting from increased dimensionality, only on CF1, all algorithms achieve the solution. However, a close look reveals that, even on CF1, only SP-UCI succeeded in obtaining function values lower than 10^{-6} (the termination criterion), whereas the results of PSO are all above 10^{-4}, and results of DE are all larger than 10^{0}. In addition, seven PSO runs were trapped by a local minimum of CF1. On CF5, only

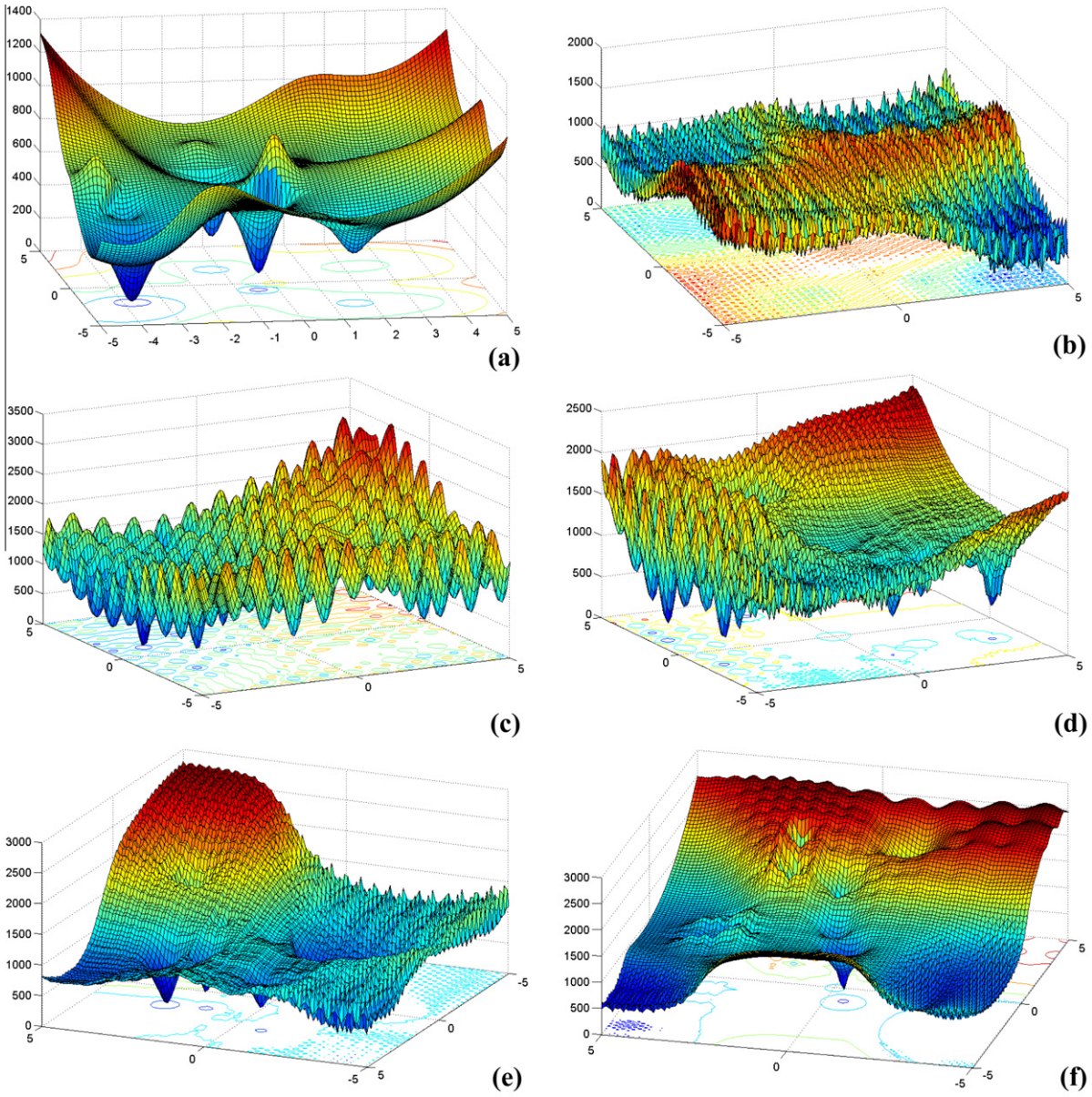

Fig. 5. Response surfaces of benchmark functions in 2-D. (a) through (f) correspond to CF1–CF6.

SP-UCI runs all converged to a small region around the global minimum, whereas runs of the other two were still farther away. On CF2 and CF3, even though no run got close to the solution, SP-UCI still achieved the best results. Finally, for CF4 and CF6, the high dimensionality made them so intimidating that all runs were trapped by the minima of the 10th basic functions, which are at the origin.

From this array of results, we can clearly see that the randomization-based algorithm, DE, only works well on low-dimensional problems. As dimensionality increases, the slope-based algorithm, SP-UCI, exhibits its persistence and yields better performances. Meanwhile, the fact that, for 100-D problems, only the simplest CF1 function can be solved supports our argument that, in high-dimensional space, only problems with well-informative landscape are solvable. CF2–CF6 all inherit more deceptiveness and randomness than does CF1 and, therefore, all three algorithms fail on them.

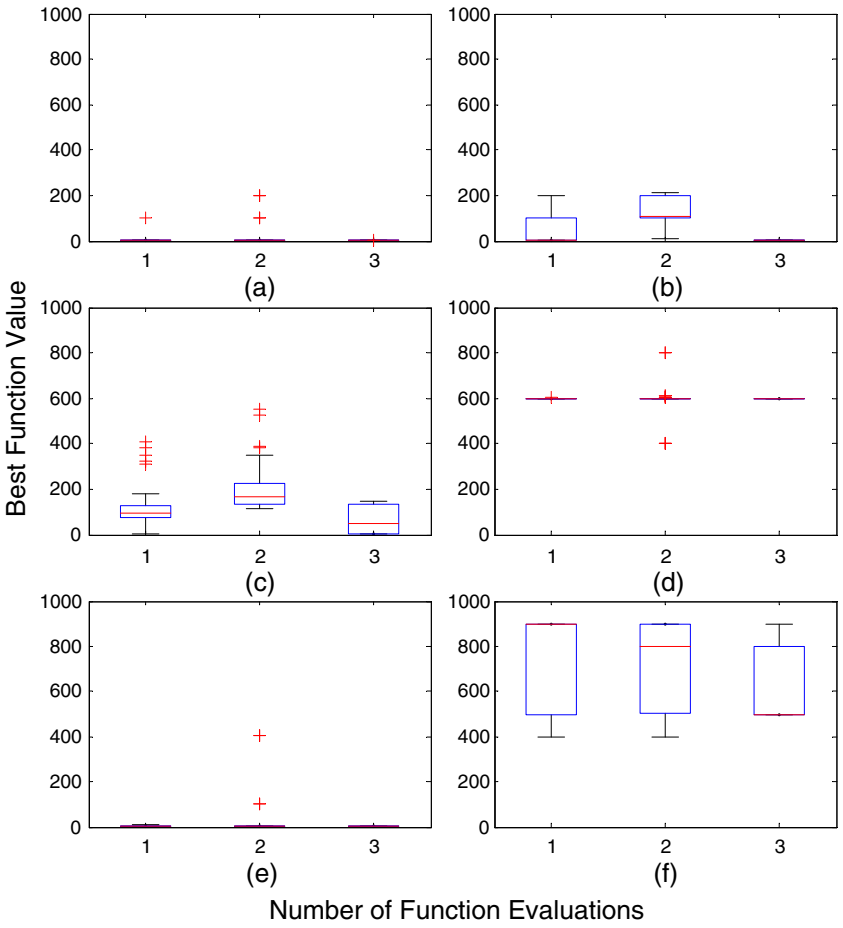

Fig. 6. Box plots of results achieved by 30 runs of 1-SP-UCI, 2-PSO, and 3-DE on 10-D benchmark functions. (a) through (f) correspond to CF1-CF6 functions. (The box has lines at the lower quartile, median, and upper quartile values. The whiskers extend to the most extreme data points not considered outliers. Outliers (denoted by +) are data with values beyond 100 units of interquartile range).

4.2. Comparison of efficiency on 100-D problems

As we mentioned, efficiency is a key issue in many real-world applications, especially for high-dimensional problems. As illustrated in Fig. 9, the results from 100-D experiments demonstrate SP-UCI's parsimoniousness in function evaluation. Owing to its sufficient exploitation of fitness landscapes, SP-UCI is able to evolve in a very efficient manner. Convergence rates of SP-UCI were strikingly higher than those of DE and PSO. SP-UCI also demonstrated its robustness by very consistent performances: Across all of the randomly initialized runs on the same test function, SP-UCI always showed very similar evolution speed and converged to the same point or a small region, whereas for PSO and DE, they sometimes had very diverse behaviors resulting from the random initialization.

CF1

Both SP-UCI and DE succeeded in consistently converging to the global optimum. However, SP-UCI converged with a much faster speed. After less than 10^5 function evaluations, the best function value (BFV) had dropped below 10^{-6}, whereas the BFV of DE runs only dropped to the order of 10^0 after 10^6 function evaluations. PSO runs converged to both the global minimum and a local minimum. For those successful PSO runs, their convergence rate was between the DE and SP-UCI runs.

CF2

Twenty-seven out of 30 SP-UCI runs converged to the minimum point of the second basic function (BFV = 100) with 3 runs converging to the minimum point of the fourth basic function (BFV = 300). Every run converged to its final point with less than 10^5 function evaluations. PSO runs spread to more local minima, and DE runs all ended at different final points spreading over the range of 150–300.

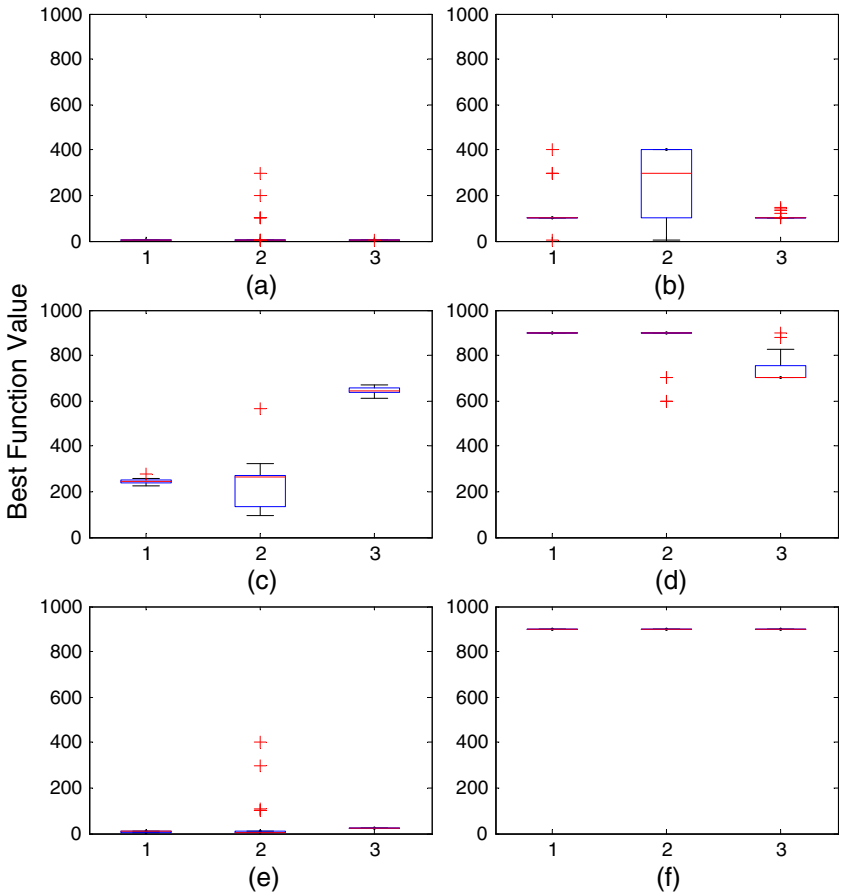

Fig. 7. Box plots of results achieved by 30 runs of 1-SP-UCI, 2-PSO, and 3-DE on 50-D benchmark functions. (a) through (f) correspond to CF1–CF6 functions. (The box has lines at the lower quartile, median, and upper quartile values. The whiskers extend to the most extreme data points not considered outliers. Outliers (denoted by +) are data with values beyond 100 units of interquartile range).

CF3

No algorithm had all of its runs converge to the same point. After around 2×10^5 function evaluations, SP-UCI runs stopped with BFV over the range of 150–350. BFV of the DE runs dropped very slowly and fell within the range of 800–950 by the end. PSO runs diverged into 2 groups again: the first group evolved even slower than DE and had final BFVs higher than 1100. The other group evolved faster than DE and had final BFV in the range of 250–600.

CF4

For every method, the difference between curves of individual runs is small. All SP-UCI runs evolved very fast, but converged to the origin. All PSO runs also converged to the same point, but with a much slower rate. Finally, DE evolved so slowly that its final BFVs only dropped to around 950 at the end.

CF5

On this function, SP-UCI runs all swiftly converged to a small region around the global minimum with mean BFV of 13.2. All DE runs evolved slowly and steadily and, by the end, they all reached a bigger region around the global minimum with BFVs around 90. PSO runs ended up with BFVs falling into three groups. The best group had an average BFV of 18.58.

CF6

In Fig. 9, plot (f) resembles plot (d). Due to the highest complexity among the six test functions, three methods were all trapped to the local minimum of the 10th basic function, whose attractive region dominated over the whole search space. SP-UCI reached to the final point and terminated quickly, whereas PSO and DE both required many more function evaluations to converge to the same point.

On all of the test functions, SP-UCI consistently displayed the highest convergence or evolution speed. DE was always the slowest one, and PSO generally behaved in between. As discussed in the introduction, this difference comes from the difference in efficiency of slope-based and random-based offspring-generating schemes. The 10- and 50-D experimental results demonstrate the similarly pattern.

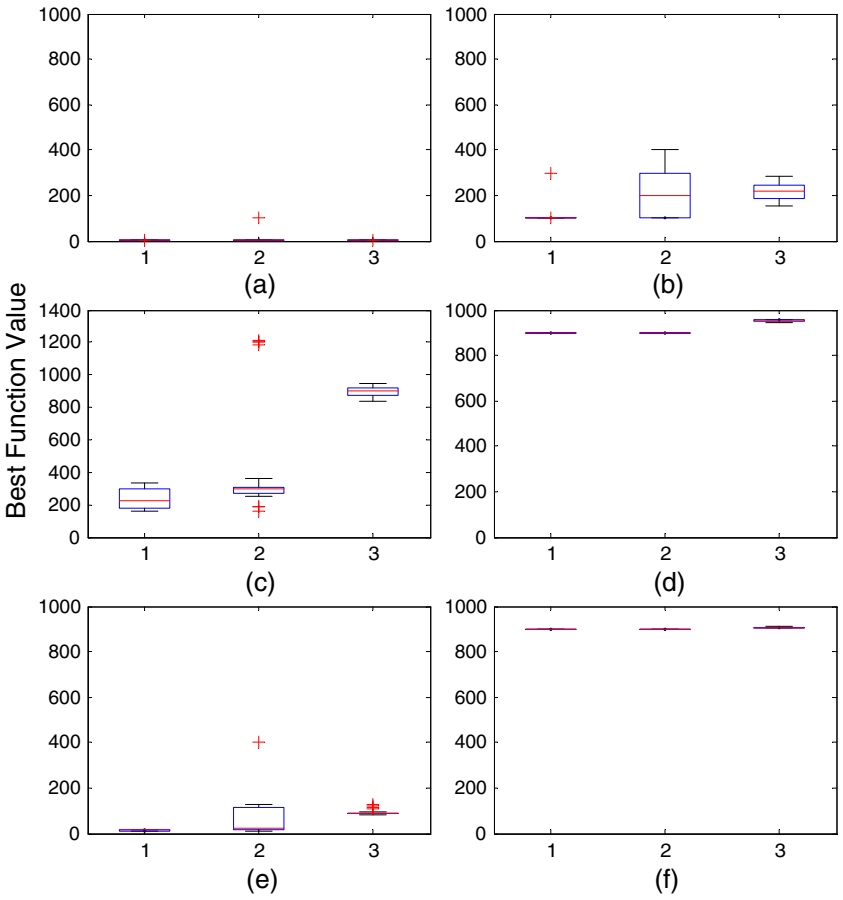

Fig. 8. Box plots of results achieved by 30 runs of 1-SP-UCI, 2-PSO, and 3-DE on 100-D benchmark functions. (a) through (f) correspond to CF1–CF6 functions. (The box has lines at the lower quartile, median, and upper quartile values. The whiskers extend to the most extreme data points not considered outliers. Outliers (denoted by +) are data with values beyond 100 units of interquartile range).

4.3. Change of difference in efficiency with increasing dimensionality

We are also very interested in how these algorithms will behave in even much higher-dimensional spaces, such as 1000-D. However, due to computational constraints, we only tested on CF1, which was the only solvable problem in 100-D. Furthermore, there were only three runs for each algorithm, since we allowed the maximum function evaluation to be 10^8, which is a extremely large number for this sophisticated composition function. Fortunately, the 100-D results show that behaviors of the three algorithms are consistent on CF1, and there is no need to have too many runs. The results are plotted in Fig. 10. Similarly, SP-UCI converged with remarkable speed. In all three runs, BFV drops to lower than 1 after 3×10^6 function evaluations. PSO also converged, but with much slower speed. By the end, which means after 10^8 function evaluations, BFV only approached close to 1 in all runs. As for DE, it was so slow that all three runs finally approached a BFV of only 900 finally.

Clearly, the 1000-D results indicate that the difference in efficiency between these three algorithms increases substantially with dimensionality. To better illustrate this, efficiency comparisons are plotted in Fig. 11 for experimental results on 10-, 50-, 100-, and 1000-D CF1 functions. Data points represent results from the median run in each case. For 10-D, the *Y*-axis represents the number of function evaluations required by each algorithm to converge to the solution. For, 50- and 100-D, since DE was not able to evolve the BFV to lower than 10^{-6}, the *Y*-axis represents the number of function evaluations that SP-UCI, PSO, and DE required to achieve final BFV of the DE run. For 1000-D, only PSO and SP-UCI were compared, and the *Y*-axis represents the number of function evolutions that SP-UCI and PSO needed to achieve the final BFV in the PSO run. This plot reveals that SP-UCI was generally faster than the other two by the order of 10 times; this advantage increases with problem dimensionality. For the 1000-D CF1 function, SP-UCI was 34 times faster than PSO and definitely faster than DE by a much larger magnitude.

4.4. Comparison with a PSO variant for high-dimensional problems with local search schemes

To further demonstrate its efficiency and effectiveness on high-dimensional problems, we also compared SP-UCI with a recent and sophisticated variant of PSO, dynamic multi-swarm particle swarm optimizer with local search for large scale

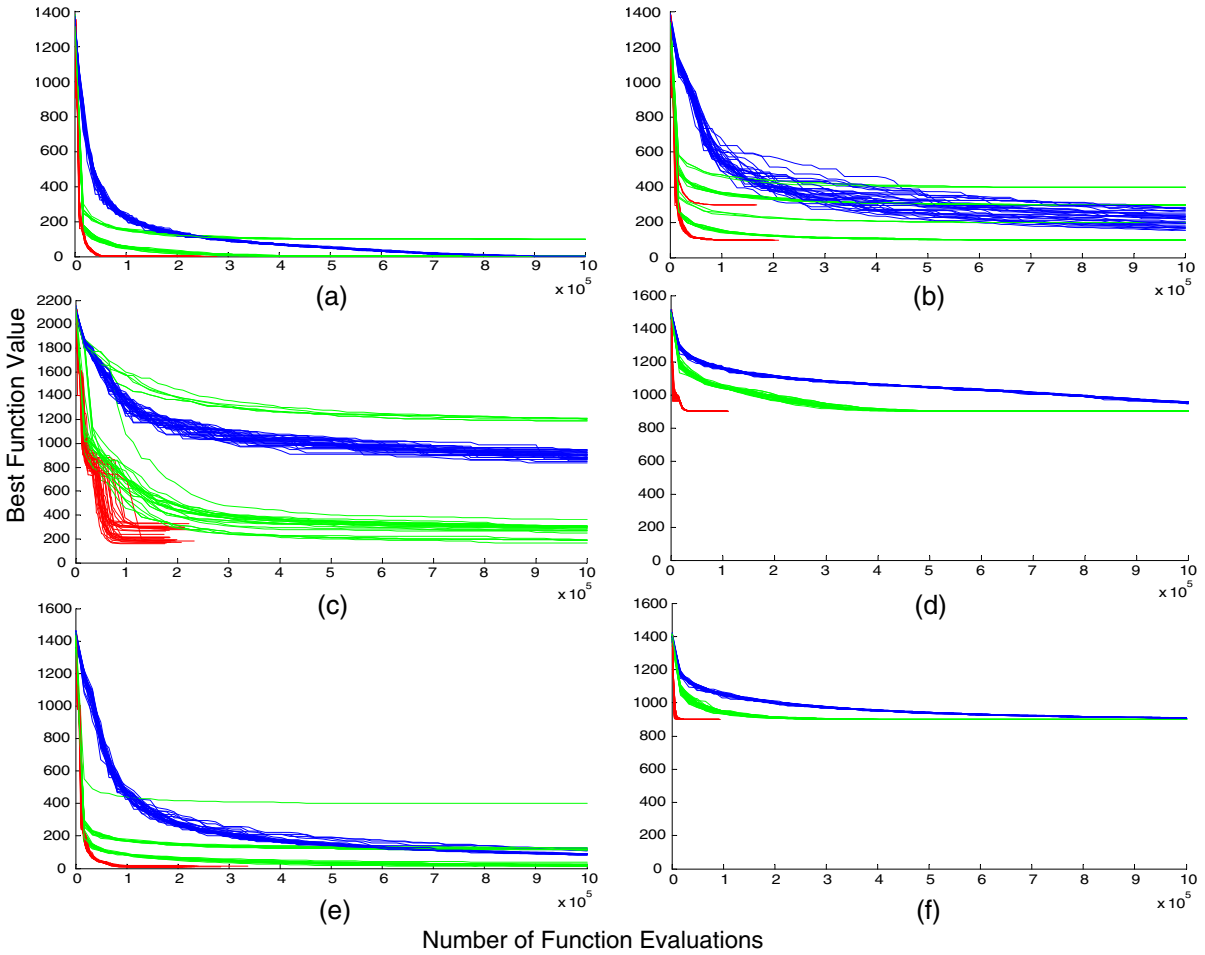

Fig. 9. Fitness curves of 30 runs of SP-UCI (red), PSO (green), and DE (blue) on 100-D benchmark functions. (a) through (f) correspond to CF1–CF6.

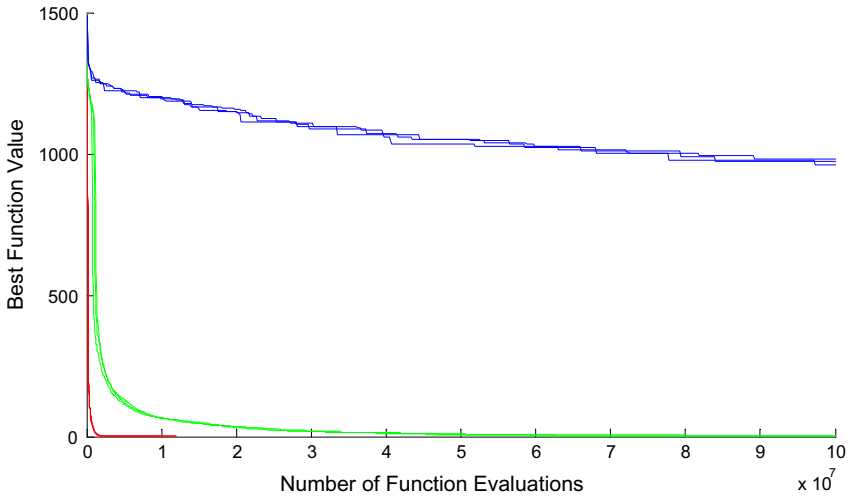

Fig. 10. Fitness curves of 3 runs of SP-UCI (red), PSO (green), and DE (blue) on 1000-D benchmark function CF1.

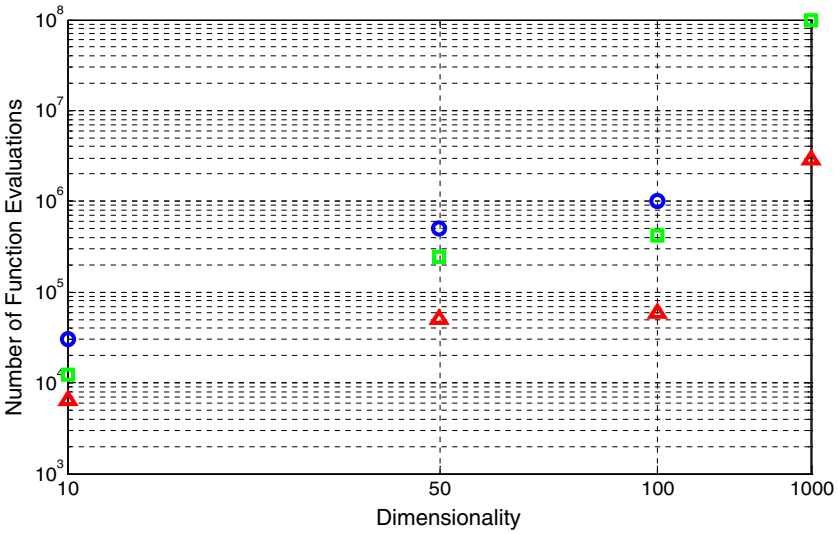

Fig. 11. Efficiency comparison of SP-UCI (triangle), PSO (square), and DE (circle) on benchmark function CF1 of 10-, 50-, 100-, and 1000-D. For each case, the point is plotted with result of the median run. The *Y*-axis represents the number of function evaluations needed to converge (for 10-D) or to reach to a small BFV (for 50-, 100-, and 1000-D). See text for details.

Table 1
Comparison of final best values (mean ± standard deviation) retrieved by SP-UCI and DSM-L-PSO.

	CF1	CF2	CF3	CF4	CF5	CF6
SP-UCI	0 ± 0	120.02 ± 61.02	239.94 ± 57.64	900 ± 0	13.21 ± 2.76	900 ± 0
DSM-L-PSO	3.33 ± 18.3	190.03 ± 106.23	767.45 ± 472.53	918.15 ± 5.53	31.81 ± 30.81	900 ± 0
P-value[a]	5.57e−10	3.99e−9	4.22e−4	1.21e−12	1.21e−10	N/A

[a] *P*-values are from Wilcoxon rank sum tests with H0: that there is no difference between the results of the two algorithm.

global optimization (DMS-L-PSO), which was proposed by Zhao et al. [36]. Different from the standard PSO, DMS-L-PSO divides the population into a large number of dynamic sub-swarms which are regrouped frequently with various regrouping schedules. The Quasi-Newton method is integrated into DMS-L-PSO to improve its local searching ability.

DMS-L-PSO code was obtained from its authors and was applied with the same settings specified in Section 3.2.1. Results are presented in Table 1, where the mean and standard deviation of final BFV from 30 independent runs are listed. On CF1 to CF5, SP-UCI always yields smaller mean and variance of BFV compared with DMS-L-PSO, whereas on CF6, the two algorithms consistently give the same results. Wilcoxon rank sum tests were conducted to test the significance of the comparison. The null assumption is H0: that there is no difference between the results of the two algorithms. The extreme small *P*-values in Table 1 substantiate that SP-UCI significantly outperforms DMS-L-PSO on CF1–CF5.

Furthermore, Fig. 12 illustrates that SP-UCI always exhibits higher efficiency compared with DMS-L-PSO.

5. Discussions

5.1. Efficiency and effectiveness

Experimental results in Section 4 serve as good examples to support our arguments regarding difficulties of high-dimensional problems and effectiveness of randomization-based and slope-based schemes. In the 10-D experiments, DE demonstrated its outstanding potency by succeeding on four problems and outperforming the other two algorithms. This indicates that randomization has the power to solve low-dimensional deceptive and random functions. However, with increasing dimensionality, the difficulty of the deceptiveness and randomness grows, and power of randomization drops. As a result, fewer benchmark functions could be solved, and the performance of DE deteriorated dramatically. The failure of DE on 1000-D CF1 shows how vulnerable randomization is to high dimensionality. On the other hand, SP-UCI excelled in high-dimensional experiments and exhibited impressive convergence rate and evolution speed. Moreover, the fact that SP-UCI achieved better or at least equal final BFVs compared with PSO and DE in the 100-D experiments indicates that slope-based schemes do have better or, at least, comparable effectiveness on high-dimensional problems.

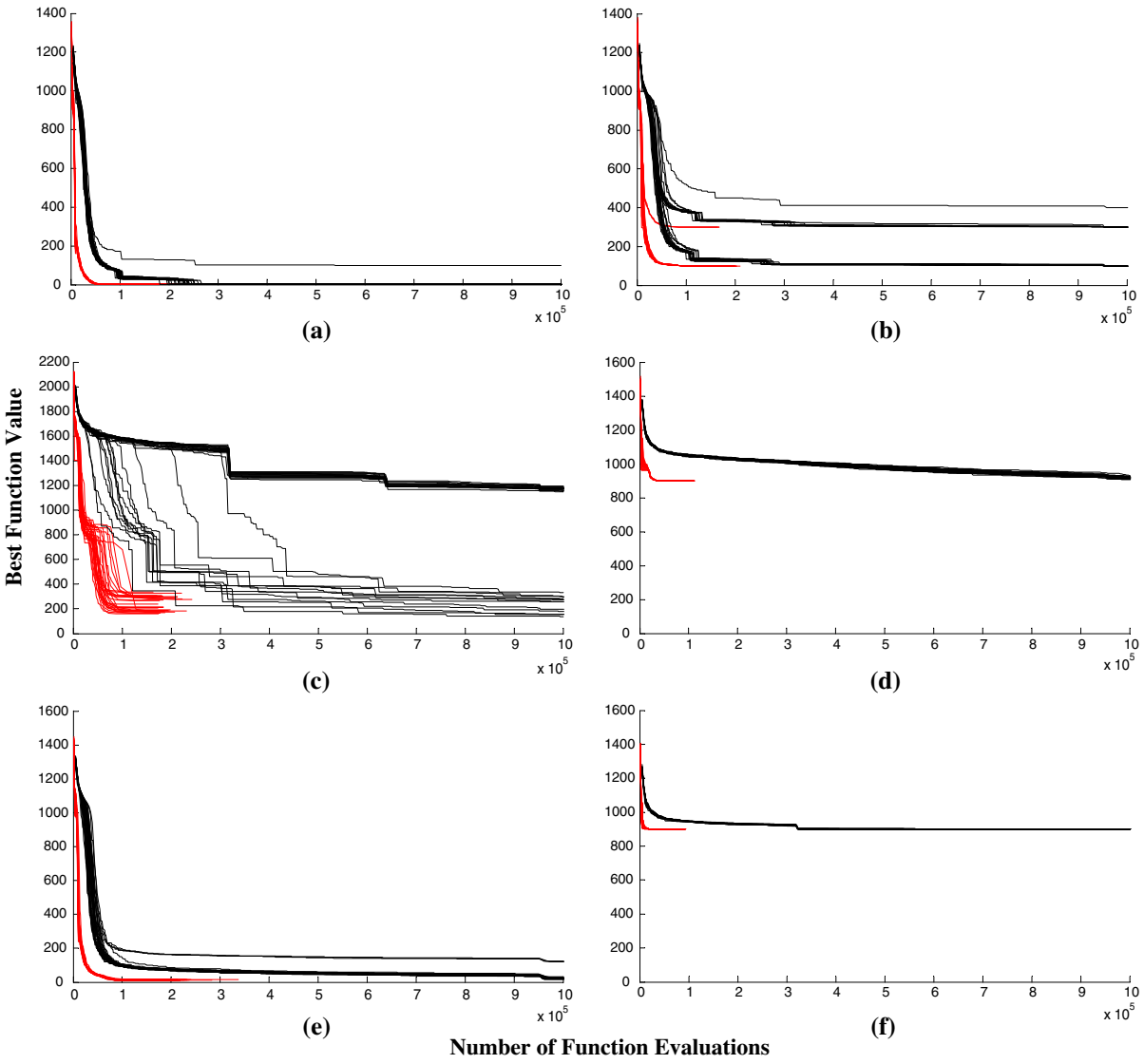

Fig. 12. Fitness curves of 30 runs of SP-UCI (red/light) and DMS-L-PSO (black/dark) on 100-D benchmark functions. (a) through (f) correspond to CF1–CF6. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

5.2. Flexibility

Meanwhile, we must admit that, in many situations, efficiency and effectiveness are mutually repulsive. Based on users' demands, a method must be able to place a proper balance. This is the reason why we built SP-UCI with four independent modules. This architecture gives sufficient flexibility to the method. By tuning each module, users can adapt the method to their expectations. Here, we briefly explain how to adjust each module and how the adjustment will affect the optimization process:

(1) In the shuffling complex scheme, the number of complexes can be changed. Increasing the number of complexes will increase the chance of finding the attractive regions of global optima, but will drag down the efficiency of evolution.
(2) For dimension restoration, by default, we set that no dimension should be neglected in every loop. However, given certain type of response surface (such as symmetric functions), SP-UCI can achieve more efficiency while the population degeneration occurs. Therefore, in this situation, we do not need to be restricted to no-lost dimensions. Instead, we can let the scheme restore the dimensionality only when more than a pre-specified number of dimensions have been lost. This pre-specified number can be used to balance efficiency and effectiveness.

(3) Within the MCCE module, we can vary the maximum number of simplex iterations. A larger number of simplex iterations leads to higher convergence rates. However, if the response surface is rough, increasing the number of simplex iterations will increase the probability of converging to local minima.
4) Multinormal resampling has the strength of overcoming local roughness, but with sacrifice of efficiency. For rough response surfaces, the more points that are resampled, the less likely that the population will be trapped by local minima. By changing the number of sampling iterations, users are able to achieve a balance given their motivations.

The above discussion is a primary introduction. Analytical and quantitative studies of the relationships between algorithmic parameters of each module and the algorithm performance will be among the focuses of our further research. The eventual goal is to make all modules self-adaptive. Nonetheless, if a user prefers, SP-UCI can always be used as a fixed method with all of the default algorithmic parameter values as we did in the experiments presented in this study.

6. Conclusions

We develop a sophisticated search algorithm, the SP-UCI method, which employs slope-based simplex strategy as well as complex and multinormal sampling and is resistant to population degeneration. Compared with two popular and more randomization-based methods, PSO and DE, this method presents salient convergence or evolution speed on high-dimensional problems. On the other hand, the fact that SP-UCI also outperforms PSO and DE by better or at least equal final BFVs in high-dimensional experiments indicates that SP-UCI does not gain efficiency simply by sacrificing effectiveness. Actually, via integrating four potent modules, SP-UCI is trying to balance effectiveness and efficiency at the same time. The capability of capturing the global minimum or improving the objective function value in a very fast manner promises SP-UCI a wide range of real-world problems.

Experimental results bear out the argument that, in high-dimensional space, only problems with well-informative fitness landscapes are solvable, and slope-based schemes are preferable to randomization-based schemes. However, we are not against the implementation of randomizations in evolutionary computation. Instead, the shuffled complex and multinormal resampling schemes in the SP-UCI method demonstrate the power of randomization in escaping local minima and sweeping over roughness response surface. In fact, the point that we are trying to make is that, in high-dimensional spaces, slope-based schemes have more potential to effectively and efficiently drive the sample population evolve compared with randomization-based schemes and should be used in constructing the kernel of searching algorithms for high-dimensional problems.

Acknowledgments

The authors thank Dr. P. N. Suganthan for kindly providing codes of the composition benchmark functions (CFs), PSO and DMS-L-PSO algorithms. The Matlab codes for DE algorithm were downloaded from Dr. R. Storn's website, and the authors also would like to express thanks to him. We also thank Corrie Thies for proofreading the manuscript. This research was supported by UCOP program of University of California (Grant 09-LR-09-116849-SORS), CPPA program of NOAA (Grants N080AR4310876 and NA050AR4310062) and NEWS program of NASA (Grant NNX06AF93G).

Appendix A

Detailed descriptions of four modules in the SP-UCI algorithm.

A.1. The main routine (the complex shuffling scheme)

The main routine of SP-UCI implements the complex shuffling scheme and drives each complex through the remaining modules sequentially:

Step 0. Initialization: in the search space, randomly sample m (number of complexes) $\times$ p (number of points in each complex) points and evaluate function at each point. In the absence of prior information, uniform sampling distribution will be used.
Step 1. Randomly and evenly divide the sample points into m complexes and sort each complex based on its corresponding finesses (function value).
Step 2. Sequentially drive each complex through modules (b), (c), and (d).
Step 3. Congregate points from all complexes and check the stop criteria. If any of the criteria are satisfied, the main routine terminates. Otherwise, return to *Step* 1.

Stop criteria are subjective and depend on users' demands. In this study, we define the following general criteria:

1) The number of function evaluations exceeds the maximum number allowed.
2) The population has converged into a pre-specified small geometric space.
3) The best function value of all sample points has not improved significantly after a pre-defined number of loops.

A.2. Dimension monitoring and restoration

During every loop of evolution, each complex first goes through module (b) to prevent from population degeneration:

Step 1. Check the dimensionality of the space spanned by all points in the complex using the following procedure:

(1) Let C be the matrix with the coordinates of each point as its columns. Then, C has the size of $d \times p$, where d is the dimensionality of the problem, and p is the number of points in each complex

$$C = [c_{ij}] = [\vec{x}_1 \dots \vec{x}_p]$$

where vectors $\vec{x}_i,\ i = 1 \dots p$, are points in the complex.

(2) Transform the original coordinated system to a normalized coordinated system by centering and normalizing each row of C and get C':

$$C' = \left[c'_{ij}\right] \quad \text{and} \quad c'_{ij} = (c_{ij} - \bar{c}_i)/\sqrt{v_i}$$

where $\bar{c}_i$ and v_i are the mean and variance of the ith row of C, respectively. Then, $\vec{x}'_j = \left[c'_{1j} \dots c'_{dj}\right]^T$ is the normalized coordinate of point j. This normalization can reduce the effect of differences in the units of different parameters in real problems. The following operations of this module are all discussed as in this normalized space.

(3) Calculate the covariance matrix of C' and denote it as R. Obtain eigenvectors and eigenvalues of R. Each eigenvector is a principal component (PC) of the complex, and its corresponding eigenvalue measures the variance of the points in the complex along the direction defined by that PC.

(4) By examining eigenvalues, we can determine if there is any dimension lost and, if yes, how many are lost. Theoretically, the complex should fully span the d-dimensional parameter space, which means that the points should have comparable variance along directions defined by every PC. If the variance along the direction of one PC is too small, it means that the complex does not span well over that direction, and that dimension is lost. On a lost dimension, we can use the centroid of the complex, $\vec{c}'$, to represent all of the points since they have very small variance on this dimension. In a d-dimensional space, for an isotropic particle population, the expected variance along each PC is $1/d$ of the total variance. Therefore, in this study, if a PC has variance less than 10% of the expected variance, we treat it as a lost dimension.

Step 2. Search along lost dimensions. For each lost dimension detected in *Step* 1, do the following random search along the PC that represents it:

(1) Sample a point from the positive side of $\vec{c}'$ along the PC.

$$\vec{x}' = \vec{c}' + ar\,\vec{l}$$

where a is a random number generated from normal distribution with mean = 2 and variance = 1, $\vec{l}$ is the unit vector representing the PC, and r is the radius of complex, defined by

$$r = \max(r_i), \quad i = 1 \dots d, \quad \text{with } r_i = \max\left(\left|c'_{ij} - c'_{ik}\right|\right), \quad j,k = 1 \dots p$$

Transform $\vec{x}'$ back to the original coordinates and evaluate the function at it. If the function value is smaller than that of the worst point in the complex, replace the worst point with this new point, and the search on this PC is over. Otherwise, discard this new point and continue to (2).

(2) Sample a point from the negative side of $\vec{c}'$ along the PC.

$$\vec{x}' = \vec{c}' - ar\,\vec{l}$$

Again, transform $\vec{x}'$ back to the original coordinates and evaluate the function at it. If the function value is smaller than that of the worst point in the complex, replace the worst point with this new point. Otherwise, discard this new point. The search on this PC terminates.

In summary, *Step* 2 is designed to quickly explore over lost dimensions to see if there is evident slope along them. If there is, the random sampling is likely to capture it, and the new point mingled into the complex will enable the complex to search along this lost dimension.

A.3. Modified competitive complex evolution (MCCE)

In the interest of high-dimensional problems, we amend the original competitive complex evolution (CCE) strategy in [9].

Step 0. Initialize index $i = 1$ and set the maximum number of iterations in this module $I = d + 1$.

Step 1. Sort the complex in order of increasing function value. Assign a triangular probability to each point, except the first one:

$$\rho_i = \frac{2(p+1-i)}{p(p+1)}, \quad i = 2,\ldots,p \quad \text{and} \quad p \text{ is the number of points in a complex.}$$

Step 2. Select the first point and d (problem dimensionality) other points from the complex according to ρ_i. Record each point's position in the complex.

Step 3. The $d+1$ selected points form a simplex, S. Generate an offspring from S by the following procedure:

(1) Sort points in S and denote them as $\vec{s}_1,\ldots,\vec{s}_{d+1}$ with the corresponding function values $f_1 < \cdots < f_{d+1}$. Calculate the centroid of the best d points and label it with $\bar{\vec{s}}$.

(2) Reflect: Reflect the worst point $\vec{s}_{d+1}$ around $\bar{\vec{s}}$ to generate a candidate point $\vec{s}_r$.

$$\vec{s}_r = 2\bar{\vec{s}} - \vec{s}_{d+1}$$

Evaluate the function at $\vec{s}_r$ and get f_r. If $f_1 < f_r < f_d$, set the offspring $\vec{s}_o = \vec{s}_r$ and go to (7).

(3) Expand: If $f_r \leqslant f_1$, reflect $\bar{\vec{s}}$ around $\vec{s}_r$ to get the extension point $\vec{s}_e$

$$\vec{s}_e = 2\vec{s}_r - \bar{\vec{s}}$$

If $f_e < f_r$, let $\vec{s}_o = \vec{s}_e$ and go to (7); otherwise, $\vec{s}_o = \vec{s}_r$ and go to (7).

(4) Contract outside: If $f_d \leqslant f_r < f_{d+1}$, calculate the outside contraction point,

$$\vec{s}_{oc} = \bar{s} + 0.5(\bar{s}_r - \bar{s})$$

If $f_{oc} < f_r$, let $\vec{s}_o = \vec{s}_{oc}$ and go to (7); otherwise, $\vec{s}_o = \vec{s}_r$ and go to (7).

(5) Contract inside: If $f_{d+1} \leqslant f_r$, calculate the inside contraction point,

$$\vec{s}_{ic} = \bar{s} + 0.5(\bar{s}_{d+1} - \bar{s})$$

If $f_{ic} < f_{d+1}$, let $\vec{s}_o = \vec{s}_{ic}$ and go to (7); otherwise, continue to (6).

(6) Multinormal sampling: If there is no better point found after going through the above simplex operations, a point will be randomly drawn with a multinormal distribution defined by the simplex. This point will replace $\vec{s}_{d+1}$ regardless of the function value at it. Let S be the matrix with every point in the simplex as its columns, and then calculate its covariance matrix, R_s. Take out the diagonal of R_s,

$$\vec{d} = \text{diagonal}(R_s) = [r_{11},\ldots,r_{dd}]$$

Modify $\vec{d}$,

$$\vec{d}' = 2(\vec{d} + \text{mean}(\vec{d}))$$

Generate a new covariance matrix R'_s with $\vec{d}'$ as its diagonal and zeros everywhere else. Finally, we can sample an offspring $\vec{s}_o$ with multinormal distribution

$$\vec{s}_o \sim \mathcal{N}\left(\bar{\vec{s}}, R'_s\right)$$

(7) Replace $\vec{s}_{d+1}$ with $\vec{s}_o$ and put the simplex back to the complex. Let $i = i+1$. If $i \leqslant I$, go to *Step* 1; otherwise, sort the complex and return to main routine.

In our study, we discern that the shrink step of the original Nelder–Mead simplex method jeopardizes the algorithm's ability to escape local minima, even though it can largely increase the convergence rate. Consequently, we excluded the shrink step from our algorithm. Meanwhile, we included the expansion and outside contraction, which are not in the CCE strategy. The random sampling operation is also different from that in CCE. We adopted the simplex-guiding multinormal distribution instead of uniform distribution. In constructing the R'_s, we keep only the diagonal of R_s in order to achieve decorrelation. Adding the mean to d' is in order to reduce the condition number of R'_s, whereas multiplying by two is to increase the sampling range.

A.4. Multinormal resampling

Population-driven multinormal resampling proves to be a powerful tool to overcome local roughness. To mitigate the influence of noisy or rough response surface, multinormal resampling is applied to the complex as well. The operation is very similar to operation (6) in *Step* 3 of module (c). The difference is that this resampling happens at the complex level. Additionally, we do not modify the covariance matrix, and p (the size of complex) new points are drawn. Note that this resampling can be repeated for I (a pre-defined number) times, depending on users' need. In this study, we set I = 1.

Appendix B

Brief descriptions of PSO and DE algorithms.

B.1. Particle swarm optimizer (PSO)

Each particle's position $\vec{x}_i$ is updated by trying a displacement (velocity) based on three sources: (1) the particle's velocity in the previous evolution $\left(\vec{v}_i'\right)$, (2) the particle's best ever position $(\hat{x}_i)$, and (3) the population's current best position $(\hat{g})$,

$$\vec{v}_i = c_0\vec{v}_i' + c_1\vec{r}_1 \otimes (\hat{x}_i - \vec{x}_i) + c_2\vec{r}_2 \otimes (\hat{g} - \vec{x}_i)$$
$$\tilde{x}_i = \vec{x}_i + \vec{v}_i$$

where c_0, c_1, and c_2 are the significance coefficients, and $\vec{r}_1, \vec{r}_2 \in R^m$ are random vectors with uniformly-generated components $\{r_p = U(0,1)\}_{p=1}^m$. $\otimes$ denotes the element-by-element vector multiplication. If $f(\tilde{x}_i) \leqslant f(\vec{x}_i)$, $\vec{x}_i = \tilde{x}_i$, otherwise, no replacement. A particle's velocity is bounded by the maximum velocity V_{max}.

B.2. Differential evolution (DE)

In the DE algorithm, for a particle $\vec{x}_i$ the candidate offspring $\tilde{x}_i$ has hybridized components $\{\tilde{x}_{ip}\}_{p=1}^m$ from $\vec{x}_i$ and another random variable $\vec{v}_i$ according to a randomly-generated number $\vec{r} = U[0,1]$:

$$\vec{v}_i = \vec{x}_a + F(\vec{x}_b - \vec{x}_c), \quad (0 < F \leqslant 2)$$
$$\vec{x}_a, \vec{x}_b, \vec{x}_c \in \{\vec{x}_j\}_{j=1}^{NS} \quad (a, b, c \neq i)$$

$$\tilde{x}_i = \left(\tilde{x}_{ip} = \begin{cases} v_{ip} & \text{if } r \leqslant C \\ x_{ip} & \text{if } r > C \end{cases}\right)_{p=1}^m$$

where $C < 1$ is the crossover constant, F is the scaling factor, and p is the component index. If $f(\tilde{x}_i) \leqslant f(\vec{x}_i)$, $\vec{x}_i = \tilde{x}_i$, otherwise, the parent survives to the next generation.

References

[1] R.E. Bellman, Dynamic Programming, Princeton University Press, Princeton, New Jersey, 1957.
[2] R.E. Bellman, Adaptive Control Processes, Princeton University Press, Princeton, New Jersey, 1961.
[3] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Transactions on Evolutionary Computation 10 (6) (2006) 646–657.
[4] J.F. Chang, P. Shi, Using investment satisfaction capability index based particle swarm optimization to construct a stock portfolio, Information Sciences 181 (14) (2011) 2989–2999.
[5] W. Chu, X. Gao, S. Sorooshian, Bound handling strategy is critical to the success of particle swarm optimization on high-dimensional complex problems, Information Sciences (2011), doi:10.1016/j.ins.2011.06.024.
[6] W. Chu, X. Gao, S. Sorooshian, A solution to the crucial problem of population degeneration in high-dimensional evolutionary optimization, IEEE Systems Journal (2011), doi:10.1109/JSYST.2011.2158682.
[7] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 526–553.
[8] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, Information Sciences 178 (15) (2008) 3096–3109.
[9] Q. Duan, V.K. Gupta, S. Sorooshian, Shuffled complex evolution approach for effective and efficient global minimization, Journal of Optimization Theory and Application 76 (3) (1993) 501–521.
[10] Q. Duan, S. Sorooshian, V.K. Gupta, Effective and efficient global optimization for conceptual rainfall-runoff models, Water Resources Research 28 (4) (1992) 1015–1031.
[11] N. Hansen, Compilation Results 2005 CEC Benchmark Function Set, 2006, pp. 1–14. (Available: <http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/compareresults.pdf>).
[12] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation, in: Proceedings of IEEE International Conference on Evolutionary Computation, Nagoya, Japan, 1996, pp. 312–317.
[13] C.T. Kelley, Detection and remediation of stagnation in the Nelder–Mead algorithm using a sufficient decrease condition, SIAM Journal on Optimization 10 (1) (1999) 43–55.
[14] J. Kennedy, R.C. Eberhart, Swarm Intelligence, Morgan Kaufman Academic Press, San Matteo, California, 2001.
[15] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983) 671–680.
[16] S. Klein, M. Staring, J. Pluim, Evaluation of optimization methods for nonrigid medical image registration using mutual information and b-splines, IEEE Transactions on Image Processing 16 (12) (2007) 2879–2890.
[17] J.C. Lagarias, J.A. Reeds, M.H. Wright, P.E. Wright, Convergence properties of the Nelder–Mead simplex method in low dimensions, SIAM Journal on Optimization 9 (1) (1998) 112–147.
[18] W.B. Langdon, R. Poli, Evolving problems to learn about particle swarm optimizers and other search algorithms, IEEE Transactions on Evolutionary Computation 11 (5) (2007) 561–578.
[19] K.Y. Lee, M.A. El-Sharkawi (eds.), Modern Heuristic Optimization Techniques with Applications to Power Systems: IEEE Power Engineering Society (02TP160), 2002.
[20] J.J. Liang, P.N. Suganthan, K. Deb, Novel composition test functions for numerical global optimization, in: Proceedings of IEEE Swarm Intelligence Symposium, 2005, pp. 68–75.
[21] K.I.M. McKinnon, Convergence of the Nelder–Mead simplex method to a nonstationary point, SIAM Journal on Optimization 9 (1) (1999) 148–158.

[22] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, may be better, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 204–210.
[23] N. Noman, H. Iba, Accelerating differential evolution using an adaptive local search, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 107–125.
[24] J. Pillardy, C. Czaplewski, A. Liwo, J. Lee, D.R. Ripoll, R. Kazmierkiewicz, S. Oldziej, W.J. Wedemeyer, K.D. Gibson, Y.A. Arnautova, J. Saunders, Y.J. Ye, H.A. Scheraga, Recent improvements in prediction of protein structure by global optimization of a potential energy function, Proceedings of National Academy of Sciences USA 98 (2001) 2329–2333.
[25] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Transactions on Evolutionary Computation 12 (1) (2008) 64–79.
[26] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.
[27] S. Sorooshian, Q. Duan, V.K. Gupta, Calibration of rainfall-runoff models: application of global optimization to the sacramento soil moisture accounting model, Water Resources Research 29 (4) (1993) 1185–1194.
[28] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC-2005 Special Session on Real-Parameter Optimization, Nanyang Technol. Univ., Singapore, 2005, pp. 1–50.
[29] M. Thyer, G. Kuczera, B.C. Bates, Probabilistic optimization for conceptual rainfall-runoff models: a comparison of the shuffled complex evolution and simulated annealing algorithms, Water Resources Research 35 (3) (1999) 767–773.
[30] P.K. Tripathi, S. Bandyopadhyay, S.K. Pal, Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients, Information Sciences 177 (22) (2007) 5033–5049.
[31] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 225–239.
[32] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 243–259.
[33] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.
[34] S.Y. Yuen, C.K. Chow, A genetic algorithm that adaptively mutates and never revisits, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 454–472.
[35] M. Zhang, W. Luo, X. Wang, Differential evolution with dynamic stochastic selection for constrained optimization, Information Sciences 178 (15) (2008) 3043–3074.
[36] S.Z. Zhao, J.J. Liang, P.N. Suganthan, M.F. Tasgetiren, Dynamic multi-swarm particle swarm optimizer with local search for large scale global optimization, in: Proceedings of IEEE Congress on Evolutionary Computation (CEC), 2008, pp. 3845–3852.